

# Modeling the Fault Tolerant Capability of a Flight Control System: An Exercise in SCR Specification\*

Chris Alexander  
Azimuth Inc.  
1000 Technology Drive  
Fairmont, WV 26554  
chrisa@azimuthwv.com

Vittorio Cortellessa  
Comp. Sc. and Electr. Eng.  
West Virginia University  
Morgantown, WV 26506-6109  
vittorio@csee.wvu.edu

Diego Del Gobbo  
Mech. and Aerosp. Eng.  
West Virginia University  
Morgantown, WV 26506  
delgobbo@cemr.wvu.edu

Ali Mili †  
Comp. Sc. and Electr. Eng.  
West Virginia University  
Morgantown, WV 26506-6109  
amili@csee.wvu.edu

Marcello Napolitano  
Mech. and Aerosp. Eng.  
West Virginia University  
Morgantown, WV 26506  
mnapolit@cemr.wvu.edu

## Abstract

In life-critical and mission-critical applications, it is important to make provisions for a wide range of contingencies, by providing means for fault tolerance. In this paper, we discuss the specification of a flight control system that is fault tolerant with respect to sensor faults. Redundancy is provided by analytical relations that hold between sensor readings; depending on the conditions, this redundancy can be used to detect, identify and accommodate sensor faults.

## Keywords

Flight Control Systems; Fault Tolerance; Flight Dynamics; Sensor Failure Detection, Identification, and Accommodation.

## 1 Introduction

Providing the fault tolerant capability (FTC) to control systems is a major issue in domains where system fault occurrences may give rise to unrecoverable damages to people and/or to very expensive devices (e.g., nuclear plants, space missions, aircrafts). In this paper we discuss modeling and specifying the fault tolerant capability of a flight control system (FCS), with respect to sensor faults. Relevant issues include: achieving fault tolerance for FCS's, defining fault domain (i.e., specifying fault hypotheses),

adequate modeling, and adequate representation of the model.

A typical approach to introduce fault tolerance in a control system is physical redundancy of components. The detection/identification of a fault is achieved by comparing the behavior of replicated components accomplishing the same task and having the same features. Cost and complexity considerations led recently an increasing interest in alternative approaches, mostly based on analytical redundancy. Outputs of components measuring different but related items are observed in order to detect/identify the faulty component. We go towards the specification of the fault tolerant capability (based on analytical redundancy) for a FCS, bounded to sensors faults.

We only focus on *critical* sensors, i.e. sensors measuring modes of the aircraft that change too quickly to be controlled by the pilot. We neglect multiple, transient and simultaneous faults. Therefore the goal of this work is not producing specifications for a complete fault tolerant capability of a real world FCS, but conceptually addressing most of issues that also persist in large scale systems. The space of sensor readings is partitioned, under fault hypotheses, and for each partition analytical relations among system variables are introduced in order to characterize the partition and to express constraints that must be satisfied when a fault occurs while system conditions fall in the partition, in order to guarantee stability and maneuverability of the aircraft.

The formulation of such relations using the Software Cost Reduction (SCR) notation is based on the tabular representation of variable behavior in SCR: it is straightforward to introduce the expression whose result is the

\*This work is supported by a grant from NASA's Dryden Flight Research Center.

† Correspondence author.

value that a variable must assume, under given conditions. Functional dependency among tables is exploited to catch out indirect relations. On the other hand, several representation/execution issues are raised here on the usage of SCR for such a domain (e.g., modeling time).

Section 2 gives an overview of a FCS, in terms of hardware/software components and input/output variables. In Section 3 analytical relations are introduced that describe how analytical redundancy provides fault tolerant capabilities to a FCS; domain partition is also provided. In Section 4 major issues related to the SCR modeling are dealt, and the specification refinement process, as part of validation, is also sketched. In Section 5 a wider perspective of the problem is given, as part of an ongoing project, where current and future possible directions are outlined. Conclusions are reported in Section 6.

## 2 A Fault Tolerant Flight Control System

### 2.1 Structure of a Flight Control System

Figure 1 shows the basic architecture of a Fly-By-Wire Flight Control System (FBW-FCS). In FBW technology conventional mechanical controls are replaced by electronic devices coupled to a digital computer. The net result is a more efficient, easier to maneuver aircraft. Four subsystems form the core of such FCS's. The *Measurement Subsystem* (MS) consists of the *Sensors* and the *Conditioning Electronics*. It measures quantities that allow observation of the state of the aircraft. *Primary* sensors are those sensors whose correct operation is required to maintain a safe flight condition. The *Actuator Subsystem* (AS) consists of the *Control Surfaces*, the *Power Control Units* (PCU's), and the *Engines*. It produces aerodynamic and thrust forces and moments by means of which the FCS controls the state of the aircraft. The *Control Panel Subsystem* contains all control devices and displays through which the pilot maneuvers the aircraft. The *Flight Control Software subsystem* (FCSw) includes all software components of the FCS. It interfaces to the hardware of the FCS through A/D and D/A cards (not shown in the figure). Current measurements, pilot inputs, and commands to the actuators are processed according to the Flight Control Law (FCL) to obtain the commands to the actuators at the next time step. Dash blocks and arrows represent the system providing Analytical Redundancy based Fault Tolerant Capability (AR-FTC) to the FCS and will be described in the next section.

### 2.2 Deploying Redundancy for Fault Tolerance

Any hardware or software fault within the FCS can compromise the safety of the aircraft. For this reason FBW-FCS's must meet strict Fault Tolerance (FT) requirements. The standard solution adopted to achieve fault tolerance is physical redundancy. A typical multichannel architecture for the FCS consists of three intercommunicating FCS's, that are equivalent —yet able to work independently. A voting mechanism checks for consistency and can, under some conditions, identify faulty components. Brute force physical redundancy is no panacea, however: product redundancy (duplicating copies of the same product) does not protect against design faults, and design redundancy (independent designs) has two major drawbacks, which are cost and complexity. Complexity, in turn, adversely affects overall reliability, and defeats the whole purpose of the fault tolerant scheme. This additional complexity affects not only the development costs, but also the maintenance costs.

These factors have, in recent years, led to an increased interest in alternative approaches for enhancing FCS's reliability. In the past two decades a variety of techniques based on *Analytical Redundancy* (AR) have been suggested for fault detection purposes in a number of applications [12]. The AR approach is based on the idea that the output of sensors measuring different but functionally related variables can be analyzed in order to detect a fault and identify the faulty component. Furthermore, preserved observability allows estimating the measurement of an isolated (allegedly faulty) sensor, while preserved controllability allows controlling the system with an isolated (allegedly faulty) actuator. Fault tolerance is achieved by means of software routines that process sensor outputs and actuator inputs to check for consistency with respect to the analytical model of the system. If an inconsistency is detected, the faulty component is isolated and the flight control law is reconfigured accordingly. By introducing AR it is possible to take off redundant sensors, electronics, mechanical linkages, hydraulic lines, PCU's, etc., thus cutting costs and weight, and reducing overall complexity of the FCS. Physical redundancy would be required only where either post-failure system observability and controllability are not preserved or detection of the fault by means of AR is not feasible in the first place.

Application of AR in FCS's is not new. The very same airplane used to conduct research on FBW technology was also used as testbed for an AR based fault detection algorithm [14]. The algorithm showed adequate performance during flight tests. However, poor robustness to modeling errors and the amount of required modeling hampered further development. Since then, a number of results have been obtained in the area of robust fault de-

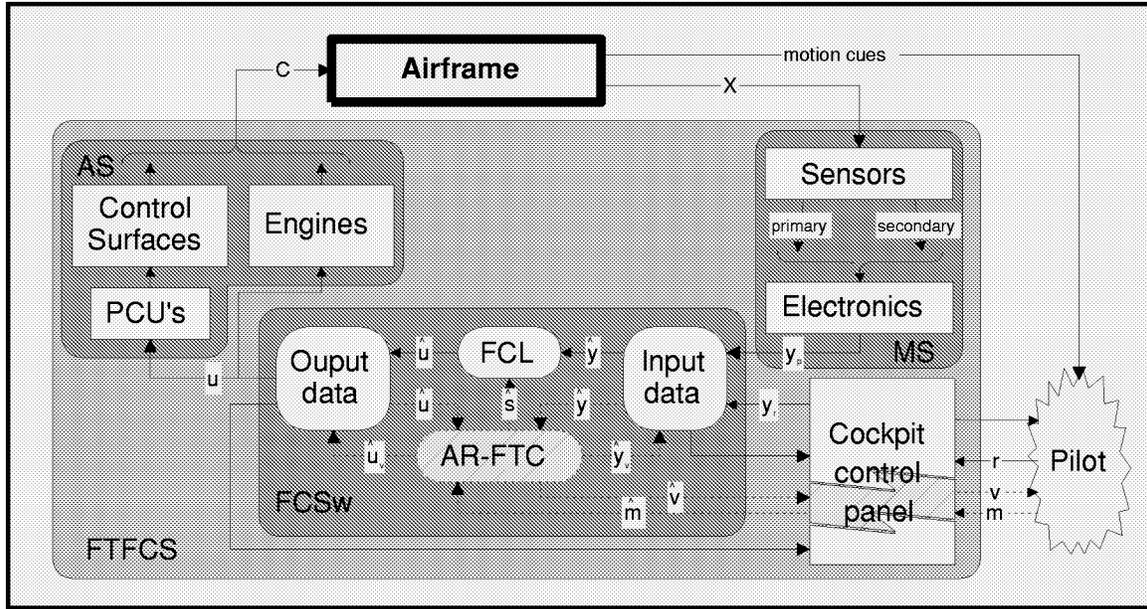


Figure 1: Fly by Wire Flight Control System.

tection [11]. *Unknown-input observers, robust parity relations, adaptive modeling, and  $H_\infty$  optimization* are a few examples. While recent research has enabled us to gain new insights into modeling analytical redundancy, it has fallen short of an integrated design methodology involving feasibility analysis, requirements specification, and certification of AR based fault tolerant control systems. Exploring strengths, weaknesses, related degree of reduction of physical redundancy, and overall reliability is a fundamental step in the engineering process of such systems.

### 3 Analytical Redundancy for Fault Tolerance

#### 3.1 The Flight Control System and Its Environment

The airplane we adopt in this study is an F16. A detailed non-linear model of the dynamics of this airplane is presented in [13]. The analytical redundancy of a fault tolerant flight control system depends on an analytical model of the system and its environment. The dynamics of many systems can be described in terms of a set of relations among its inputs, outputs, states, and state derivatives. These relations represent constraints imposed by laws of mechanics, electronics, and thermodynamics upon system inputs, outputs, and their derivatives. Because of neglected dynamics, disturbances, and measurement errors

the analytical model of a system is not necessarily a truthful representation of the real system. Control system designers are mindful of this discrepancy and adopt design techniques that are robust with respect to such uncertainties.

In describing how analytical redundancy provides fault tolerance capabilities to a flight control system, we adopt the following notation:

$$R_c(\text{state, state derivative, input, output, process uncertainty, measurement error}) \quad (1)$$

$$R_d(\text{current state, new state, input, output, process uncertainty, measurement error}) \quad (2)$$

Parameters involved in these relations are vectors. The relations are deterministic with respect to the first four parameters, and stochastic with respect to the last two. Whenever a parameter is not involved in a relation, we write the symbol '-' in its place. Relation (1) is used for time *continuous* models, where all parameters are evaluated at the same instant  $t$ . Relation (2) is used for time *discrete* models, where all parameters are evaluated at the discrete time  $t_k$ , except the *new state*, which is evaluated at  $t_{k+1}$ . The difference  $(t_{k+1} - t_k)$  is the sampling time of the discrete system. For the sake of simplicity we describe only the most relevant parameters of the relations that we introduce in the remainder of the paper. Accordingly, we will often skip the description of *state*, *state-derivative*, *process-uncertainty*, and *measurement-error* parameters unless they play a prominent role in the analytical redun-

dancy framework.

The following relations represent the analytical models of the hardware systems in Fig. 1:

$$P(x(t), \dot{x}(t), c(t), x(t), \xi(t), -) \quad (3)$$

$$A(x_a(t), \dot{x}_a(t), u(t), c(t), \xi_a(t), \eta_a(t)) \quad (4)$$

$$M_p(x_p(t), \dot{x}_p(t), x(t), y_p(t), \xi_p(t), \eta_p(t)) \quad (5)$$

$$M_r(x_r(t), \dot{x}_r(t), r(t), y_r(t), \xi_r(t), \eta_r(t)) \quad (6)$$

Relation (3) describes the dynamics of the aircraft, i.e. the *process* to be controlled by the FCS. It involves force, moment, kinematics, and navigation equations. The state vector includes the flight variables used in the above equations. A typical state vector is:

$$x = [U, V, W, P, Q, R, \Phi, \Theta, \Psi, p_N, p_E, h]^T \quad (7)$$

where the elements are the three linear velocities, the three angular velocities, the three attitude angles, and North, East, and altitude position of the aircraft. Forces and moments applied to the airframe by the control surfaces and the engines are included in the input vector  $c(t)$ . The output coincides with the aircraft state and represents the *actual* value of the flight variables in (7). Since it is a set of actual values, there is no output uncertainty. The process error vector  $\xi$  is related to the uncertainty of the relation with respect to neglected dynamics and unknown inputs.

Relation (4) describes the dynamics of the actuator subsystem. The input vector  $u(t)$  includes command signals to the actuators, while the output vector includes thrust and aerodynamic forces and moments. Relation (5) describes the dynamics between aircraft state  $x(t)$  and process measurements  $y_p(t)$ . A typical set of sensors provides the following measurements:

$$y_p = [P_s, P_t, \alpha, \beta, A_x, A_y, A_z, \tilde{P}, \tilde{Q}, \tilde{R}]^T \quad (8)$$

These are static pressure, total pressure, angle of attack, sideslip angle, body accelerations, and body angular rates respectively. To differentiate *measured* from *actual* body rates we adopt the 'tilde' notation. Relation (6) describes the dynamics between the *actual* position of pilot controls  $r(t)$  and their measurements  $y_r(t)$ . The two measurement vectors will be referred to in the sequel as:

$$y(t) = [y_p(t), y_r(t)]^T \quad (9)$$

The following relations represent the analytical models of the software systems in Fig. 1:

$$I(-, -, y(t_k), \hat{y}(t_k), -, \eta_I(t_k)) \quad (10)$$

$$L(x_L(t_k), x_L(t_{k+1}), [\hat{y}(t_k), \hat{u}(t_k)], \hat{u}(t_{k+1}), -, \eta_L(t_k)) \quad (11)$$

$$O(-, -, \hat{u}(t_k), u(t_k), -, \eta_O(t_k)) \quad (12)$$

The FCSw closes the control loop between sensors and actuators subsystems. To distinguish software variables from related electrical signals we adopt the 'hat' notation. Relation (10) represents the relationship between measurement samples  $y(t_k)$  and the corresponding software variables  $\hat{y}(t_k)$ . Since this is an algebraic relation, there is no need to introduce state variables.  $\eta(t_k)$  takes into account quantization error. Relation (11) describes the dynamics of the the flight control law. Current value of sensor measurements and actuator commands are processed to produce the actuator commands at the next time step  $\hat{x}(t_{k+1})$ . Relation (12) describes the relationship between software commands  $\hat{u}(t_k)$  and electrical commands  $u(t_k)$ .

In order to complete the set of relationships needed to illustrate the principles at the basis of AR-FTFCS's we introduce two relationships capturing the FT requirements:

$$R_h(x(t), \dot{x}(t), r(t), \dot{r}(t)) \quad (13)$$

$$R_l(x(t), \dot{x}(t), r(t), \dot{r}(t)) \quad (14)$$

Relation (13) describes the high priority responsiveness requirements of the aircraft to pilot commands in terms of the true state of the airplane, the input commands, and their derivatives. Relation (14) describes the low priority requirements. For an airplane to be safe it is mandatory that the high priority requirements are preserved even in case of fault.

### 3.2 The AR-FTFCS

After having introduced the analytical model of the FCS, its environment, and its fault tolerant requirements it is possible to illustrate how an AR-FTFCS works.

At the instant  $t_k$  new measurements are available as software data  $\hat{y}(t_k)$ . The elements of this vector are not independent; they are correlated by means of relations (3), (5), (4), (10), and (12). Furthermore, they are correlated to  $\hat{u}(t_k)$  by virtue of the same relations. By analyzing sensor measurement and actuator command histories it is possible to check whether the above relations are satisfied. If a fault within the hardware loop produces an inconsistency with respect to the analytical model the system is said to hold AR properties allowing *detection* of the fault. After detection of the fault it is necessary to identify which component has failed. Each component of the FCS plays a different role within relations (5) and (4). Hence, the distortion affecting these relations at the occurrence of a failure depends on the component failed and on the fault mode. By processing sensor measurement and actuator command histories it is possible to locate the source of distortion. If a fault within the hardware loop produces a distinct signature in terms of commands/measurements

correlation the system is said to hold AR properties allowing *identification* of the fault. Once the faulty component is identified, the FCS needs to be accommodated in order to preserve responsiveness requirements. Accommodation can be carried out at the software level because the flight control algorithm is not unique. Given relations (3), (5), (4), (10), and (12) describing the dynamics of the aircraft, sensors, actuators, and interfaces with the FCL, there can be a number of different control algorithms satisfying responsiveness requirements. Some of these algorithms do not use all of the sensors and/or actuators available. Hence, if a hardware component of the FCS fails, responsiveness requirements can be maintained by switching to a control algorithm that does not employ that component. If such an algorithm exists the system is said to hold AR allowing *accommodation* of the fault.

Figure 1 shows how a FCS is enhanced to an AR-FTFCS. The dash blocks and arrows represent the subsystem providing Fault Tolerant Capability (FTC) to the FCS. The core of this subsystem is the AR-FTC software module, while the dash section within the *Control Panel* block represents the hardware interface to the pilot. Following the notation adopted in the previous section we describe the FTC subsystem by means of the following two relations:

$$AR_{Sw}(x_{ar}(t_k), x_{ar}(t_{k+1}), [\hat{y}(t_k), \hat{u}(t_k), \hat{m}(t_k)], [\hat{y}_v(t_k), \hat{u}_v(t_{k+1}), \hat{v}(t_k), \hat{s}(t_k)], -, \eta_{ar}(t_k)) \quad (15)$$

$$AR_{Hw}(-, -, [\hat{v}(t_k), m(t_k)], [\hat{m}(t_k), v(t_k)], -, \eta_{ar}(t_k)) \quad (16)$$

Relation (15) describes the dynamics of the software module. It processes current measurements  $\hat{y}(t_k)$  and commands  $\hat{u}(t_k)$  to validate  $\hat{y}(t_k)$  against the analytical model of the system. If no inconsistencies are detected the FCL module takes over and produces the new command  $\hat{u}(t_{k+1})$ . If an inconsistency is detected the AR-FTC module further processes incoming data to identify the faulty component. Then, it either produces a *virtual* set of validated measurements  $\hat{y}_v(t_k)$  for the FCL, or it bypasses the FCL and produces a new set of validated commands  $\hat{u}_v(t_{k+1})$  according to a *safe* control law that does not use the faulty component. The two options are typically adopted for sensor and actuator faults respectively. If a component of the actuator subsystem fails then it is often necessary to reconfigure the control law to take into account the control deficiency. If a sensor fails its output can be estimated and the estimation substituted into the measurement vector  $\hat{y}(t_k)$  to produce  $\hat{y}_v(t_k)$ . However, the solution can be adopted where the FCL is bypassed and an alternative control law is used in its place. The  $\hat{s}(t_k)$  signal is used to synchronize execution of the FCL and the AR-FTC modules.  $\hat{m}(t_k)$  and  $\hat{v}(t_k)$  are the operational mode selected by the pilot and the diagnos-

Fault mode	$y_i$ (Volts)	$\hat{y}_i$ (deg/sec)
Loss of signal	[2.0, 2.5]	[-22, 0]
Loss of power	0	-90
Loss of ground	12	90

Table 1: Fault modes

tic information respectively. Relation (16) represents the relationship between pilot's controls  $m(t_k)$  and displays  $v(t_k)$  and related software variables  $\hat{m}(t_k)$  and  $\hat{v}(t_k)$ .

### 3.3 Fault Hypotheses

In the previous section we have shown how a system featuring AR properties can be made fault tolerant with respect to sensor faults at the software level. However, software along with its supporting hardware (computers, data buses, etc.) and hardware systems other than sensors and actuators can fail as well. AR cannot be adopted to provide fault tolerance with respect to failure of such components; a different approach must be adopted for these types of faults.

In this phase of the study we focus our attention on sensor faults only. More specifically, we require fault tolerance with respect to failure of the *roll*, *pitch*, and *yaw* rate gyros. The sensors used are a solid state rate gyro where a vibrating element is used to measure rotational velocity by employing the Coriolis principle. The output range of the sensor is  $\pm 90$ deg/sec and its bandwidth is 18Hz. The output of the sensor has been recorded while simulating the failures. Three different fault modes have been considered: *loss of signal*, *loss of power*, *loss of ground reference*. The fault modes along with outputs of the sensor and the values of the correspondent software variable are listed in Table 1.

### 3.4 Fault Modeling

We have to point out that even though analytical redundancy does enable us to achieve some level of fault tolerance, it does not guarantee arbitrary levels of precision in detecting, identifying, and accommodating sensor faults. An exhaustive feasibility analysis covering components subject to failure, fault modes, and possible state evolution for actuator, aircraft, and sensor systems is required. To explain how detectability and identifiability problems arise we will refer once again to relations (3), (5), (4), (10), and (12). These relations can be assembled to form one single relation that captures the system AR at software level in terms of sensor measurement and actuator command histories:

$$R_g([\hat{u}(t_k), \hat{u}(t_{k-1}), \dots, \hat{u}(t_{k-m})]),$$

$$\begin{aligned} & [\hat{y}(t_k), \hat{y}(t_{k-1}), \dots, \hat{y}(t_{k-n})], \\ & [\hat{\nu}(t_k), \hat{\nu}(t_{k-1}), \dots, \hat{\nu}(t_{k-p})] \end{aligned} \quad (17)$$

Here  $\nu$  represents a global uncertainty term collecting all process uncertainty and measurement error terms in relations (3), (5), (4), (10), and (12). Variables  $n$ ,  $m$ , and  $p$  represent the depths of the input, output, and uncertainty sequences respectively. If we consider the space whose points have coordinates given by the elements involved in relation (17), we can distinguish those regions of the space where relation (17) is satisfied, from those where it is not. Furthermore, we can characterize those regions related to distortions of relation (17) caused by the given fault modes.

Following Mili et al. [1, 7], we model a fault tolerant scheme by means of a partition of the relevant state space into a hierarchy of classes that represent degrees of *correctness*, degrees of *maskability*, and degrees of *recoverability*. For a program, the relevant state space is the set defined in terms of all the values taken by all the state variables of the program; for a set of sensors, the relevant state space is the set defined in terms of all the values taken by all the sensor readings. The partition that we derive for our purposes is given in figure 2. Process uncertainties (disturbances, simplifying hypotheses, modeling shortcuts, etc) make the actual partition more complex than the original model [1, 7]. In its current form, this partition is incomplete, and is being refined.

The inner ellipsis of figure 2 represents states for which the deterministic relationship within relation (17) holds. The outer ellipsis contains the points for which the stochastic relation (17) holds. Points outside this region imply an inconsistency with the analytical model of the system. The three triangles marked  $F1$ ,  $F2$ , and  $F3$  contain points related to three different fault modes. The intersection between the region ' $F1 \cup F2 \cup F3$ ' and the outer ellipsis contains those points that are related to a fault mode, but that preserve relation (17). Hence, this region represents those states where a fault mode is not detectable by means of AR. This region is marked *Detection non Feasible* in the figure. The region marked *Identification non Feasible* contains those points for which the identification of a fault cannot be achieved either because that fault mode has not been considered within the specifications, or because failure effects do not allow us to distinguish between the fault modes.

To describe accommodation feasibility in analogous terms we need to consider the space whose points represent the aircraft states. If after the failure of a component the FCL can be reconfigured to satisfy the safety requirements then accommodation is feasible within the whole space. If instead there are states that cannot be reached without violating the safety requirements the space will be partitioned in regions where accommodation is feasi-

ble and regions where it is not. As a limit case, accommodation is not feasible in the whole space if there is no FCL that would satisfy the safety requirements.

## 4 SCR Modeling

The derivation of this specification is part of a larger project whose purpose is to validate and certify an adaptive fault tolerant capability (AR-FTC in figure 1) for a flight control system, which is concurrently being implemented using a *radial base function* neural network [8]. Our intent is that the specification will be used as an oracle in the testing task which aims to validate/ certify the fault tolerant capability. Consequently, it is rather imperative that the specification be written in a language that is supported by automated tools; so that in the validation/ certification phase, the neural net and the executable specification can be executed independently to provide a basis for checking the former against the latter. We have chosen to use SCR as the specification vehicle, because it lends itself to this type of application: tabular representations, which form the semantic foundations of SCR, were used in [3] to specify the requirements of the Navy's A7-E aircraft and in [10] to specify nuclear power plants; SCR was used to specify an autopilot [2], to specify a variety of high assurance applications [5, 6], and to specify some functions of the space shuttle software [15].

### 4.1 Scope of the Specification

Figure 1 shows the structure of the overall aircraft system, including the data flow between the aircraft, the flight control system, the cockpit controls, and the environment. The first issue we must address is to delimit the boundaries of our specification. We have pondered two possible options, which we denote by option 1 and option 2: whereas option 1 focuses on the inputs and outputs of the fault tolerant capability component (re: *AR-FTC*, in figure 1), option 2 (the aggregate of the flight control system, with the aircraft) considers the impact of the outputs of the AR-FTC on the aircraft state. The choice of an option is driven by the following considerations.

- *Generality/ Abstraction.* For a given situation, defined by a set of sensor readings, there are many sequences of actions that a flight control system can follow to achieve/ maintain the maneuverability/ stability of the aircraft. At any instant, these actions may be different, but their combined effect over time is identical. Hence by virtue of abstraction (we do not wish to deal with the detailed mechanics of how the AR-FTC operates) and generality (writing specifications that apply across a wide range of possible

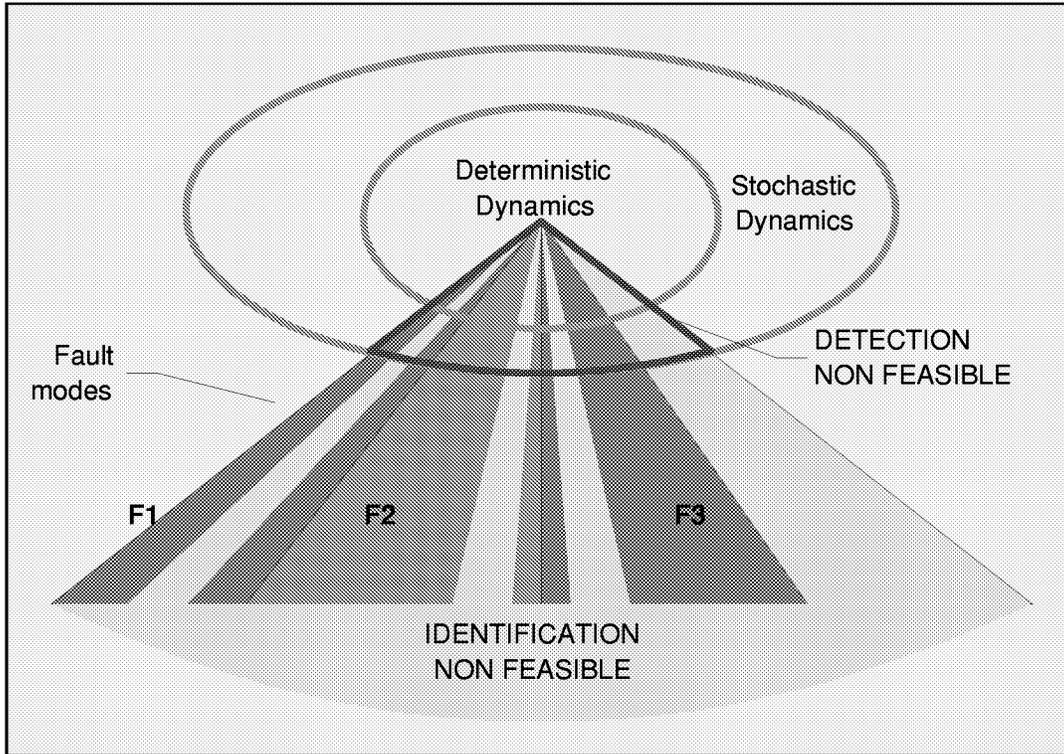


Figure 2: Partition of the Space of Sensor Readings

implementations), the second option is better than the first.

- *Observability/ Controllability.* If we choose option 2, then we cannot judge the outputs of the AR-FTC directly, but we have to observe their effect on the aircraft. This gives us much lower observability of the AR-FTC than option 1. Controllability is the same for both options.

Ultimately, this decision amounts to choosing between observability (option 1), i.e. the ability to observe and monitor the exact values that are produced by the FTFCS implementation, and abstraction (option 2), i.e. the ability to give the implementer some latitude in how to maintain maneuverability. We have ruled in favor of abstraction.

## 4.2 Representation Issues

By virtue of the choice discussed above, the input variables of the specification are the sensor readings of relevant flight parameters (altitude, speed, acceleration, angle of attack, rate gyros, aileron deflections, elevator deflections, rudder deflection) and the actuator input values; the output variables are the actual values (i.e., the validated vectors) of the same parameters and a fault report.

Hence, for parameter  $Y$ , for example, we define variable  $mY$  (named after SCR parlance: monitored  $Y$ ) which represents the sensor reading for  $Y$ , and variable  $cY$  (controlled  $Y$ ) which represents the actual value of parameter  $Y$ . Because of sensor failures,  $cY$  may differ from  $mY$ . In addition, because of the latency of the FCS and (especially) of the aircraft (in reacting to adjusted actuator values), the value of  $cY$  at time  $t$  ( $cY(t)$ ) is not functionally related to the value of  $mY$  at the same time  $t$  ( $mY(t)$ ), but rather to previous values of  $mY$ . In addition to the sensor readings of the flight parameters, the set of input variables also includes the values of the cockpit controls. The second and fourth columns of table given in figure 3 show the structure of the input and output spaces. We now give a mapping of those spaces into the partition of sensor readings in figure 2.

The innermost ellipsis of figure 2 represents the fault-free case, whose relation takes the form

$$R = \{(\langle mY, mU \rangle, \langle cY, cU \rangle) | p(mY, mU, cY, cU)\},$$

where predicate  $p$  characterizes the condition of deterministic freedom. The outermost ellipsis of figure 2 represents the case in which a measurement error in sensor readings leads to uncertainty in the fault process; the rela-

Family of Variables	Input (monitored) Variable	State (Term) Variable	Output (controlled) Variable
Flight Parameter, $Y$	$mY$	$tY$	$cY$
Actuator Inputs, $U$	$mU$	$tU$	$cU$
Pilot Command, $MR$	$mMR$		
Fault Report, $V$			$cV$

Figure 3: Space Structure

tion for this ellipsis takes the form

$$R' = \{(\langle mY, mU \rangle, \langle cY, cU \rangle) | q(mY, mU, cY, cU, \eta, \xi)\},$$

where  $\eta$  represents the measurement error and  $\xi$  the process uncertainty; predicate  $q$  characterizes the condition of fault status in the stochastic dynamics.

Each triangle  $F_i$  of figure 2 represents a different fault mode, whose relation take the form

$$R_{F_i} = \{(\langle mY, mU \rangle) | f_i(mY, mU)\},$$

where predicate  $f_i$  characterize the condition of fault mode  $i$ . Different shades and bold lines in figure 2 separate areas with different fault capabilities. In order for a fault  $i$  to be detected, it must satisfy

$$R_{F_i} \cap R = \emptyset;$$

also, to be identified, it must satisfy

$$R_{F_i} \cap R_{F_j} = \emptyset \quad (\forall j \neq i).$$

With regard to accomodability, the specification must reflect the property that the aircraft remains maneuverable despite the presence of sensor faults. In particular, in every triangle  $F_i$  maneuverability is a binary function in variables  $mMR(t)$  and  $cY(t)$  (linking pilot commands to actual flight parameters). Because  $mMR(t)$  and  $cY(t)$  are not instant variables but rather functions of time, it is conceivable that the value of  $cY$  at some time  $t$  be a function of the value of  $mMR$  at a previous time  $t' < t$ .

### 4.3 Modeling Issues

Time is inherent in the specification of the FTFCS. Execution of the FTFCS takes place in the context of a sequence of sensor inputs which, except for faults, represents a physically feasible flight path. The FTFCS is aware of the passage of time through the advent of clock pulses;

at each clock pulse, the FTFCS takes a snapshot of the sensor readings, processes them, computes actuator values, then awaits the next clock pulse. Note that the sensor readings may well remain constant across two or more clock pulses; the FTFCS processes them at each clock pulse all the same. On the other hand, sensor readings may take several distinct values between two successive clock pulses; the FTFCS is only aware of their two values at the successive clock pulses.

Whereas the real-time operation of the FTFCS is driven by the clock pulses, the execution of the SCR specification (for the purposes of validating the specification or verifying implementations against it) is driven by the successive application of the functions defined by SCR's tabular expressions on the input variables and the state variables. SCR specifications are executed in a kind of a *batch* mode, where the real time between two successive function applications need not bear any relation to the actual time between two successive clock pulses.

The concept of time arises naturally in flight dynamics equations, which are differential equations of flight parameters and pilot commands. Let us consider some controlled variable  $cX$ , and let us assume that this variable satisfies the following differential equation:

$$\frac{d(cX)}{dt} = F(t),$$

where  $t$  is the time variable,  $F$  is a function of  $t$  that potentially involves monitored and controlled variables (including  $cX$ ), and  $\frac{dX}{dt}$  is the derivative of  $X$  with respect to time. If we approximate the derivative by means of finite differences, we find  $\frac{(cX(t) - cX(t - \delta t))}{\delta t} = F(t)$ . If we let  $\delta t$  be the interval between two successive clock pulses, and let this be the unit of time (i.e.,  $\delta t = 1$ ), then  $cX(t)$  and  $cX(t - \delta t)$  measure the current value and the past value of parameter  $cX$ . Solving this equation for  $cX(t)$ , we find

$$cX(t) = cX(t - \delta t) + F(t). \quad (18)$$

Each application of this transformation (from  $cX(t - \delta t)$  to  $cX(t)$ ) represents the effect of the advent of one clock pulse. In order to distinguish between the current and past values of variable  $cX$ , we use SCR's concept of *term variable*. To each flight parameter ( $X$ ) we associate a *term variable*, which we denote by prefixing the variable name with  $t$ ; the term variable is used to represent the value of the variable at the preceding clock pulse. In order to represent the transformation described in equation (18), we write SCR tables to perform the following transformations *in sequence*.

$$\begin{aligned} tX &:= cX; \\ cX &:= tX + F; \end{aligned}$$

We cannot merely define two tabular expressions that compute variables  $cX$  and  $tX$  according to these formulas, for they produce a circular reference (and SCR does not recognize the sequence command—the order of execution in SCR is driven merely by functional dependencies).

A tantalizing alternative is, of course, to use SCR's primed variable convention, whereby the primed version of any given (non-primed) variable is the previous value of that variable. This option does not work for our purposes, because of the specific interpretation of *previous value* in SCR. SCR is event-driven, where each change of value of any variable is understood to be an event; by contrast, our model is time-driven, where an event is the advent of a clock pulse. If, between two successive clock pulses, three monitored variables change values, SCR considers that it has witnessed three events, and *previous* refers to the most recent one; by contrast, our model considers that only one event has occurred, and *previous* refers to the state of the system at the previous clock pulse.

## 5 Assessment

In this section, we review our specification project (although it is still in progress) and assess some of our decisions, with partial hindsight.

### 5.1 SCR Adequacy

We briefly report on our experience with using SCR for the purposes of our specific situation. We acknowledge that we have very little prior experience with SCR, and our comments must be qualified accordingly.

The general pattern of a table in SCR is to compute a controlled variable in terms of monitored variables and possibly term variables. Many requirements that we encounter are instead best formulated as a relation between monitored variables (to limit the domain of a relation), or a relation between controlled variables (to limit the range of a relation). Also, even if the controlled variable is a deterministic function of the monitored variables, it may be more natural to represent this function by a conjunction of non-oriented, non-deterministic, relations.

We find it unsettling that in a specification that has a large number of tables, the only composition operator between these tables is functional dependency, which is not even explicit. We would find it more powerful to have a wide vocabulary of composition operators which we can use to compose tables together. There is undoubtedly a sound basis for letting functional dependency be the sole criterion that determines the order of evaluation of tabular expressions. But our experience with modeling time would have been more successful if we had the ability to

impose an arbitrary sequencing between tables, to break the cycle of circular dependencies. (Note: It is possible to define a refinement-monotonic sequence like operator, using demonic semantics). Furthermore, because tables are combined only with functional dependencies (rather with refinement-monotonic composition operators) we find no natural discipline for stepwise specification generation. Such a discipline would enable us to compose a specification in a stepwise manner, and to know that as we produce more and more tables, the overall specification grows increasingly more refined (until completeness). The structure afforded by such explicit composition operators can be used to control the complexity of subsequent validation and verification tasks.

Because SCR, and the tabular expressions on which it is based [9, 4], support model-based specifications, it elicits more detail from the specifier than a behavioral specification. This excess detail makes the specification more complex, and may lead to inconsistencies.

We find that the data type offerings of SCR are more akin to those of a programming language than to those of a specification language. In an application such as ours, we needed a variety of data types, ranging from angles (degrees) to durations (seconds) to engine speeds (rotations per minute) to positions (meters) to speeds (meters/second) to accelerations (meters/second/second) to angular velocity (degree/second), etc. We found ourselves mapping all of them into reals, when a language supported typing system that provided a wide range of data types and a corresponding type checking function would have enhanced the readability and reliability of our specification. We also found that it would have been helpful if SCR provided dimension-checking functions, whereby whenever we write an equation, it checks the dimensions of both sides to ensure that they are consistent. Whether this is an extension of the type checking function, a separate function, or actually the same (only more elaborate) function, we do not know.

Many of the issues that we raised here are interrelated (e.g. a single design decision dictates a host of interrelated issues), and many stem from legitimate design tradeoffs (e.g. favoring efficient executability vs expressive power). We assume that as we become more acquainted with the spirit of SCR's specification model, some of these issues will may grow increasingly insignificant.

### 5.2 Non-determinacy

We faced a dilemma while trying to derive a specification for the fault tolerant capability flight control system, dealing with the determinacy of the specification. We had two options:

*Make the specification deterministic.* This is more natural, from the standpoint of SCR (which revolves around the pattern of formulating controlled variables as a function of monitored variables), and yields generally simpler specifications. The main drawback of this solution, of course, is that it forces us to second guess the designer of the neural net, because we have to derive a specification for the exact function that the neural net is implementing. This, in turn, has two drawbacks: first, it imposes much coordination between the implementer team and the specifier team, and is counterproductive from a V&V viewpoint (V&V relies primarily on redundancy); second, it imposes early constraints on the designer, prohibiting him from altering design decisions that affect the specifier team.

*Make the specification non deterministic.* The position here is to let the specification focus on expressing the desired functional properties, without going as far as to uniquely specify which output will satisfy these desired properties. This solution is consistent with traditional guidelines for good specification, but causes some difficulty in SCR, because SCR does not handle non-determinacy naturally. This is the most striking limitation we have encountered using SCR. In our example (and certainly in many other applications as well), we often encounter requirements that are not deterministic; also, many complex requirements are best formulated as the aggregate of a set of simpler, non-deterministic requirements.

We felt very justified in choosing the second option, but have found that it raises an issue which may, with hindsight, cause us to reassess our choice: Under the first option (deterministic specification), the specification of the system does not have to capture the criteria under which differences of output between the specification and the implementation can be considered tolerable; this decision can be made during the verification and validation step, by the V&V team, to take into consideration any special circumstances that may arise at run-time. By contrast, under the second option, the tolerance margins have to be hardcoded into the specification, and cannot be adjusted subsequently by the V&V team to account for special testing/operational conditions. Hence both options force us to make early decisions: The first option imposes on us to agree with the implementer on specific design decisions; the second option imposes on us to agree with the V&V team on specific tolerance margins.

## 6 Prospects

### 6.1 A Testing Plan

Our plan calls for using the target specification as an oracle in the test plan of the neural network. Specifically, the neural net feeds its inputs into a certified flight simulator, which plays the role of the aircraft components in the graph of figure 1. This aggregate is placed side by side with the SCR specification, whereby the SCR is used as an oracle to test the neural net. Input data is submitted to the system under test and the SCR oracle, to check for correctness. This input data is the aggregate of sensor readings and pilot controls, which are collected from previously collected flight simulation data. The purpose of the testing plan is to make a ruling on the certifiability of the neural net as an implementation of the fault tolerant capability of the flight control system. The system structure that we have derived for this purpose is presented in figure 4. The fault reports of the neural net and the SCR specification are compared for logical equality, producing the result shown in the lower right corner of the figure. On the other hand, the actual state of the aircraft, produced by the flight simulator, is matched against the pilot controls (by virtue of a law that captures aircraft maneuverability), to return a boolean indicator of whether the aircraft maintains adequate maneuverability (despite the possible presence of faults).

### 6.2 Interpreting Flight Dynamics Equations

In the process of deriving the SCR specification of the FTFC system, we are really conducting two activities, namely modeling and representation:

- *Modeling.* This task deals with such matters as deciding which parameters are of interest, how do we represent the fault tolerant capability, how do we represent time, how do we approximate derivatives, how do we enforce sequencing of tabular evaluations/executions, how do we reflect the dynamic nature of the system, how do we detect, identify and accommodate faults, etc.
- *Representation.* Generally speaking, this matter deals with how do we map our model into SCR terms, and how do we formulate our model in such a way as to take the best advantage of built-in SCR features.

Ideally, we would like to think of these two activities as being strictly sequential; i.e. modeling must be completed before representation can proceed. As attractive as it may

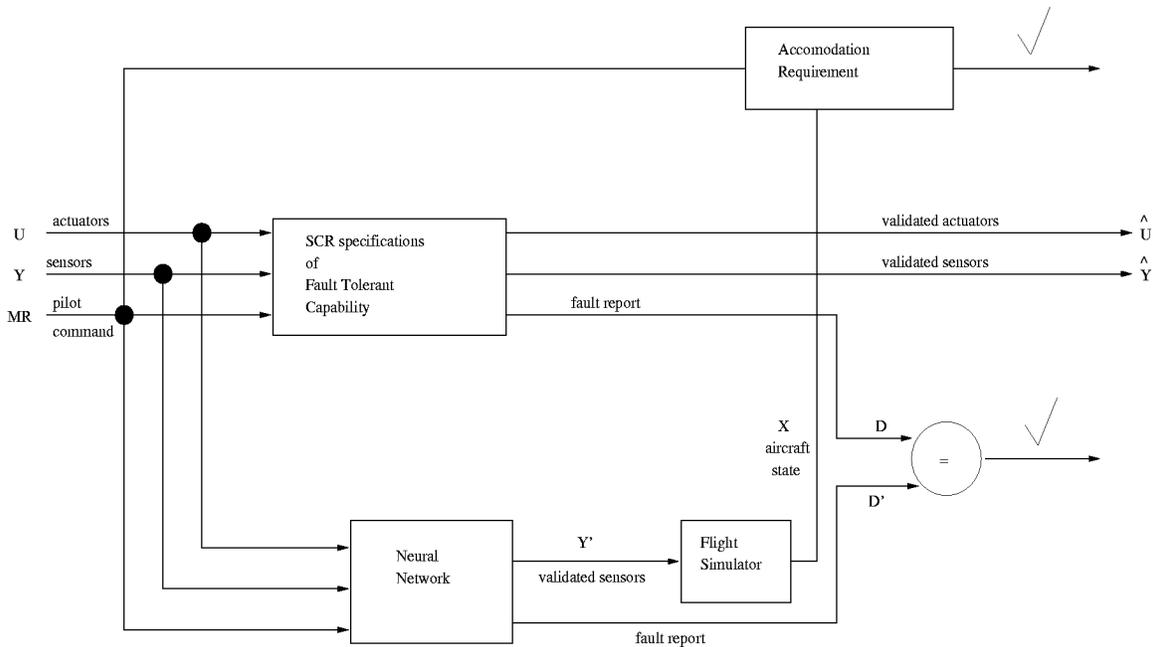


Figure 4: A Testing Plan for the Neural Net

be, this discipline has proven to be a challenge in practice, due to time pressures and to the impact of representation constraints on modeling decisions. This has led us to consider the possibility of producing a syntax directed translation of flight dynamics equations into SCR source code. The main advantage of this solution is that we get to encode all our modeling decisions in the syntax-directed rules; this allows us to keep our modeling options open until very late in the specification lifecycle; most important of all, this solution ensures that our modeling decisions are applied uniformly across all the equations of the specification.

### 6.3 Analytical Reasoning on Neural Nets

Traditional certification algorithms observe the behavior of a software product under test, and make probabilistic/statistical inferences on the operational attributes of the product (reliability, availability, etc). The crucial hypothesis on which these probabilistic/statistical arguments are based is that the software product will reproduce under field usage the behavior that it has exhibited under test. This hypothesis does not hold for adaptive neural nets, because they evolve their behavior (learn) as they practice their function. Of course, one may argue that they evolve their behavior for the better; but *better* in the sense of a neural net (convergence) is not necessarily better in the sense of correctness verification (monotonicity with respect to the refinement ordering). Concretely, a neu-

ral net may very well satisfy the SCR specification in the testing phase, and fail to satisfy it in the field usage phase, even though it converges. See figure 5.

In light of these observations, we envisage to complement the certification testing activity with an analytical method. Such a method would rely on some semantic analysis of the neural net, as well as some hypothesis regarding the data that it receives in the future.

## 7 Summary

In this paper we have discussed the formal specification, in SCR, of an adaptive fault tolerant flight control system. The specification is due to be used as an oracle in the certification of a radial basis function neural net that implements the adaptive scheme. The fault tolerant properties of the system, the adaptive nature of its implementation, and the specific application for which the specification is intended (certification), contribute to make this a unique experiment in system modeling and representation. In particular, the fact that the system implementation is adaptive (hence does not duplicate its behavior as it evolves) rules out traditional testing techniques. Also, the fact that the system's behavior is dependent on input history precludes the traditional static analysis techniques. The specification generation is under way, and we expect many of the modeling and representation decisions that we have discuss in this paper to remain *in flux*.

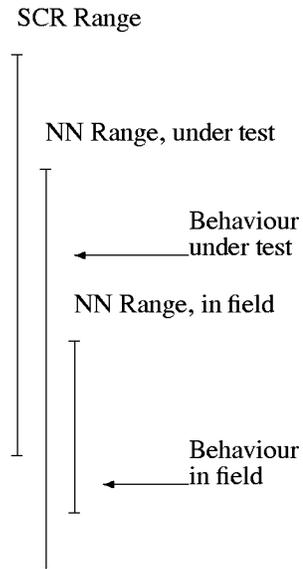


Figure 5: Convergence does Not Ensure Consistency

## Acknowledgments

The authors acknowledge the assistance of Dr Constance Heitmeyer and Dr Ralph Jeffords, from the Naval Research Laboratory, with the use of SCR. Also, we thank Prof. Steve Easterbrook, University of Toronto, who contributed his knowledge and experience with SCR.

## References

- [1] H. Ammar, B. Cukic, C. Fuhrman, and Mili. A comparative analysis of hardware and software fault tolerance: Impact on software reliability engineering. *Annals of Software Engineering*, 10, 2000.
- [2] R. Bharadwaj and C. Heitmeyer. Applying the SCR requirements method to a simple autopilot. In *Proceedings, Fourth Langley Formal Methods Workshop*, Hampton, VA, September 1997.
- [3] K. L. Heninger, J. Kallander, D. L. Parnas, and J. E. Shore. Software requirements for the A-7E aircraft. Technical Report 3876, United States Naval Research Laboratory, Washington D. C., 1978.
- [4] R. Janicki, D. L. Parnas, and J. Zucker. Tabular representations in relational documents. In Ch. Brink, W. Kahl, and G. Schmidt, editors, *Relational Methods in Computer Science*, chapter 12, pages 184–196. Springer Verlag, January 1997.
- [5] J. Kirby Jr, M. Archer, and C. Heitmeyer. Applying formal methods to a high security device: An experience report. In *Proceedings, IEEE International Symposium on High Assurance Systems Engineering*, pages 81–88. IEEE Computer Society Press, November 1999.
- [6] J. Kirby Jr, M. Archer, and C. Heitmeyer. SCR: A practical approach to building high assurance: Comsec system. In *Proceedings, Annual Computer Security Applications Conference*. IEEE Computer Society Press, December 1999.
- [7] A. Mili. *An Introduction to Program Fault Tolerance: A Structured Programming Approach*. Prentice-Hall, Englewood Cliffs, NJ, 1990.
- [8] F.E. Nasuti and M. Napolitano. Sensor failure detection, identification and accomodation using radial basis function networks. Technical report, West Virginia University, Mechanical and Aerospace Engineering, Morgantown, WV, November 1999.
- [9] D. L. Parnas. Tabular representation of relations. Technical Report 260, Communications Research Laboratory, Faculty of Engineering, McMaster University, Hamilton, Ontario, Canada, October 1992.
- [10] D. L. Parnas, G. J. K. Asmis, and J. Madey. Assessment of safety-critical software in nuclear power plants. *Nuclear Safety*, 32(2):189–198, April-June 1991.
- [11] Ron J. Patton. Robust model-based fault diagnosis: The state of the art. In T. Ruokonen, editor, *IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes: SAFEPROCESS '94*, volume 1, pages 1–24, Helsinki Univ. Technol, Espoo, Finland, June 1994. IFAC, Springer Verlag.
- [12] Ron J. Patton, Paul Frank, and Robert Clark. *Fault Diagnosis in Dynamic Systems: Theory and Applications*. Prentice Hall, 1989.
- [13] Brian L. Stevens and Frank L. Lewis. *Aircraft Control and Simulation*. John Wiley and Sons, New York, N.Y., 1992.
- [14] K.J. Szalai, R.R. Larson, and R.D. Glover. Flight experience with flight control redundancy management. In *AGARD Lecture Series No.109. Fault Tolerance Design and Redundancy Management Techniques*, AGARD Lecture Series, pages 8/1–27. AGARD, Neuilly-sur-Seine, France, October 1980.
- [15] V. Wiels and S. Easterbrook. Formal modeling of space shuttle change request using SCR. Technical Report NASA-IVV-98-004, NASA IV& V Facility, Fairmont, WV 26554, <http://www.ivv.nasa.gov/>, 1998.